# 嵌入式 linux 中 nRF24l01 驱动的设计与实现①

李　进，王太宏，张恩迪

(湖南大学 微纳光电器件及应用教育部重点实验室，长沙 410082)

**摘　要**：介绍了 nRF24l01 芯片的性能特点，并结合 linux 系统下字符设备驱动开发的流程和框架，设计了该芯片的 liunx 驱动程序。其中提出并验证了中断和巡检两种处理方式来接收和发送数据，以适应在实际应用中不同的开发需要。最后给出了基于此驱动的应用程序的设计方法。

**关键词**：nRF24L01；linux；字符设备；驱动；中断和巡检

## Design of nRF24l01 Driver under Embedded Linux System

LI Jin, WANG Tai-Hong, ZHANG En-Di

(Key Laboratory for Micro-Nano Optoelectronic Devices of Ministry of Education, Hunan University, Changsha 410082, China)

**Abstract**: This article introduces the parameters and performance of the chip nRF24l01, and designs the driver for the chip under linux system refer to the course and the structure of the char device driver development. Additionally, the article designs and verifies interrupt and polling two ways for receving and sending data to meet the different demands of the real application, and introduces the design methods for the applications based on the driver.

**Key words**: nRF24L01; linux; char device; driver; interrupt and polling

随着电子信息技术的发展，嵌入式系统以其体积小，性价比高，功耗低等特点已经越来越受到人们的重视，并得到广泛应用。但是，目前硬件设备的种类日益繁多，linux 系统不能够给它们都提供现成的驱动，研究和设计新型硬件的驱动成为了嵌入式开发的最重要的内容之一。nRF24l01 是一种新型的低耗，高速的无线通讯芯片，在消费类电子无线通讯，住宅、楼宇智能无线数据采集和控制等方面有着很广泛的实际应用，所以在 linux 环境下开发与之相应的驱动有着比较大的价值，本文根据 linux 字符设备开发的框架结构，结合该芯片的特点，给出了一种可行的设计方案。

## 1　nRF24l01芯片的主要性能介绍及相关分析

nRF24l01 是单片射频收发芯片，工作于 2.4～2.5 GHz ISM 频段。工作电压为 1．9～3．6 V。可通过 SPI 写入数据，空中数据传输率最高可达 2 Mb / s，有自动应答和自动再发射功能，通过对相应寄存器的写入和读出来完成对芯片的控制，当数据重发，发送或接收成功时，IRQ 引脚会产生下降沿的跳变（中断功能开启时），标志寄存器中的相应位也会置“1”，来提示 MCU 做相应处理[1]。

根据 nRF24l01 芯片的性能特点可以看出实现该设备的驱动关键在于以下三点：

（1）实现 SPI 总线的通讯功能。

（2）实现 RF24L01 的相应寄存器的初始化工作。

（3）识别处理数据的收发情况，确保在数据传输中不发生数据丢包的现象。

本文的驱动程序设计也主要围绕以上三点功能进行。

## 2　驱动程序的设计和实现

在 linux 系统中，一个完善的驱动程序既需要有对该硬件设备的功能支持，同时也要符合设计驱动模块的一般框架。本文在实现上首先设计一个名为“nRF24l01.h”的头文件，包含硬件连接的引脚定义, SPI

① 基金项目:973 项目(2007CB310500);国家基金委项目(90606009)
收稿时间:2011-01-04;收到修改稿时间:2011-02-28

总线的实现，各个寄存器的初始化定义等相应配置，然后编写“nRF24L01.c”文件通过调用“nRF24L01.h”以及其他系统头文件来完成驱动程序的 open，close ，read，write，ioctl 等功能函数。

### 2.1 配置文件 nRF24l01.h 的设计实现

首先需要完成各个硬件接口的定义与配置。下面是其中中断引脚定义的部分程序：（其他引脚与之类似）

```
#define IRQ            S3C2410_GPF(5)
#define IRQ_IN  s3c2410_gpio_cfgpin (IRQ, S3C2410_GPIO_INPUT)
#define IRQ_STU     s3c2410_gpio_getpin(IRQ)
```

由于驱动是在 linux 系统下基于 S3C2440 芯片的设计，在定义引脚时，需要根据 linux 不同的版本使用不同的 gpio 函数，本文设计中使用了 2.6.32 版本的内核因此在定义引脚的方向时应使用 s3c2410_gpio_cfgpin(IRQ,S3C2410_GPIO_INPUT)函数[2]。

引脚配置成功以后就需要实现 SPI 总线通讯以及寄存器的初始化工作，SPI 的通信实现采用了 IO 端口模拟的方法，然后将引脚替换为已经定义好的宏定义。在此基础上可实现一系列对 nRF24L01 寄存器操作的功能函数，最后在该头文件的末尾，根据实际过程中对通讯参数的具体要求来设计一个 init_nrf24l01 函数来实现 nRF24L01 的初始化工作，其中对 CONFIG 寄存器的配置需要注意，要根据当前芯片的工作状况来设置成不同的状态。

### 2.2 驱动源文件 nRF24l01.c 的设计与实现

结合 nRF24L01 芯片的特点以及 linux 系统下设备类型的不同，可以把它认为是字符设备的一种。本文采用了字符设备中特有的“混杂设备”驱动的方法来设计，其特点在于有固定的主设备号“10”，不需要系统来分配，另外在加载该驱动时也会自动生成设备文件，避免了手工生成设备文件的繁琐[3]。

根据 linux 字符驱动的特定架构，本文设计的 nRF24l01 芯片驱动由 open，close ，ioctl，read，write，poll 等函数结构组成，并且均被设计在 nRF24l01.c 文件当中，以完成不同的功能。

#### 2.2.1 open 与 close 函数设计

应用程序在打开和关闭设备文件时需要调用 open 和 close 函数，由于在设计当中开启了 nRF24L01 的中断功能，因此在 open 函数中要采用 linux 内核提供的中断注册函数 request_irq 来注册 nRF24L01 的 IRQ 引脚中断，同时也应该把在 nRF24L01.h 中设计的 init_nrf24l01 函数包含在 open 函数中以实现 nRF24L01 的初始化工作，另外在 close 函数中，为了释放注册过的中断资源，需要使用 free_irq 函数。

#### 2.2.2 interrupt 与 poll 函数设计

为了处理某些外设的突发性事件，linux 内核引入了中断机制来进行管理。当在内核当中注册的中断被触发时，中断函数就会被调用。在本文 nRF24l01 的驱动设计中，中断函数 nrf24l01_interrupt 用来响应由于自动重发超过限制次数，发送数据成功，接收数据成功而引起的 IRQ 引脚的下降沿跳变，其函数体主要完成区分中断类型，清除中断标志位，设置全局变量的功能。

在 linux 内核中 poll 函数用来监测文件的的状态，当文件的状态未发生变化时，在未超时的情况下它的用户态 select 函数将一直阻塞当前进程的运行，在本设计当中，该函数体用于和中断函数配合，具体实现上，当 poll 函数被调用时，它会检测由中断程序控制的全局变量，根据该值的大小来向应用程序返回当前文件的状态从而控制程序的运行，以下是 interrupt 与 poll 函数的部分主要代码：

```
static irqreturn_t nrf24l01_interrupt(int irq,void *dev_id)  // 中断函数
{     sta=SPI_Read(STATUS); // 读取标志寄存器状态
if(sta & 0x10)        //    判断是否为多次重发中断
{ int_point=1;        //    给全局变量赋值
SPI_RW_Reg(WRITE_REG+STATUS,sta);// 清 除中断标志位 }
else if  (sta & 0x20)     // 判断是否为发送成功中断
{ ....}
else if  (sta & 0x40)     // 判断是否为接收成功中断
{ ....}
return IRQ_RETVAL(IRQ_HANDLED);
static unsigned int nrf24l01_poll( struct file *file, struct poll_table_struct *wait)
// poll 函数
```

```
{   unsigned int mask;
poll_wait(file, &nrf24l01_waitq, wait);
// 加入等待队列
if(int_point==2)   // 检测全局变量
mask |= POLLOUT | POLLWRNORM; // 设置返回的文件状态为可写
else if (int_point==3)
mask |= POLLIN | POLLRDNORM   //// 设置返回的文件状态为可读
return mask ;}
```

中断和 poll 函数的配合使得应用程序只有在中断时才会向下执行，其他时候则处于阻塞状态，可以很好地识别芯片数据收发的状况，提高了内核运行效率。

2.2.3 ioctl 函数的设计

在 linux 内核中，ioctl 函数通过参数的传递来完成对硬件设备的控制，具体到本文的驱动设计当中，与 interrupt 函数功能相同，也是用来完成数据发送接收情况的识别，不过需要读取标志寄存器来实现。与中断的方法相比，应用程序需要在反复调用该函数才能实现功能，但是该函数不会阻塞进程，实现程序也相对简单。在不同的应用背景下，这两种方法各有其优势，为了使驱动本身具有更广的应用范围，笔者同时设计了这两种方法，下面是 ioclt 函数主要的功能代码：

```
static int nrf24l01_ioctl(struct inode *inode,struct file
*file,unsigned int cmd , unsigned long arg)
{   unsigned char sta;
unsigned char status;
sta=SPI_Read(STATUS);     //读取寄存器状态
switch（sta）                  // 判断寄存器状态
case 0x10: status=1,break;     //给返回值赋值
.....
return   status; }
```

从函数体可以看出，通过返回值 status 的大小，应用程序可以获悉当前数据的传输情况，由于没有中断类似的事件机制，因此无法确定数据收发成功的确切时机，在使用时需要反复调用该函数。

2.2.4 read 和 write 函数的设计以及 file_operat- ions 结构的声明

linux 内核提供了 read 和 write 函数来完成数据在应用程序和驱动之间的交互工作，在本文的驱动设计当中，有数据要发送出去时，write 函数内部首先通过 copy_from_user 函数来从用户态获取数据然后调用 nRF24L01.h 定义的 nRF24l01_TxPacket 函数，将数据发送。当需要读取数据时，则与之相反依次调用 copy_to_user 和 nRF24l01_RxPacket 函数来实现功能。

最后当主要的驱动函数模块设计完毕以后需要调用一个 file_operations 的结构体来完成总体的声明，其中包含了以上设计的各个函数的名称[4]。

## 3　驱动程序的安装

驱动程序设计完成以后，首先在内核源代码以及字符驱动的目录下，修改 Makefile 文件，做好编译前的配置工作，然后将驱动.h 文件和.c 文件移至字符驱动目录下，分别执行“make clean”，“make modules”命令后,将生成名为 nRF24L01.ko 的目标文件，最后将该文件从宿主机移至目标机上执行“insmod nRF24L01.ko”命令即完成驱动模块的加载，同时也可执行“rmmod nRF24L01”命令来卸载模块[5]。

## 4　驱动程序的验证以及应用程序的设计

由于应用程序可以采用中断或者巡检两种方法来识别数据的收发情况，为了验证各自的功能性，所以文章采用了中断的方法来发送数据，巡检的方法来接收数据。具体实现上，发送数据时，首先执行 write 函数然后调用 select 函数，当系统接收到有效中断信号时，select 函数将取消阻塞状态，再来执行下一次的发送工作。接收数据时，与之类似通过 ioctl 函数来不断地获得寄存器的状态然后再来调用 read 函数接收数据。以下是各自的主体代码架构：

中断方式：

```
main( )
{   int nrf24l01_fd,i, receive[32];
   fd_setrds;
nrf24l01_fd=open("/dev/nrf24l01",0);   //以默认方式打开设备文件
write(nrf24l01_fd,&receive[i],1);  // 调用写函数
while(1)
{ ret=select(button_fd+1,NULL,&rds,NULL,NULL);//调用 select 函数,阻塞进程
i++;
write(nrf24l01_fd,&receive[i],1);   //发生中断取消
```

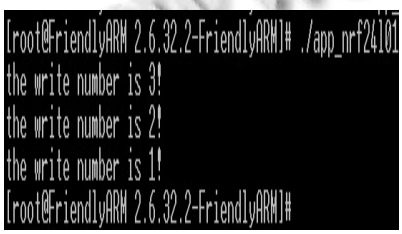
阻塞后，写入下一部分数据

```
    .....
}
close( nrf24l01_fd); }
```

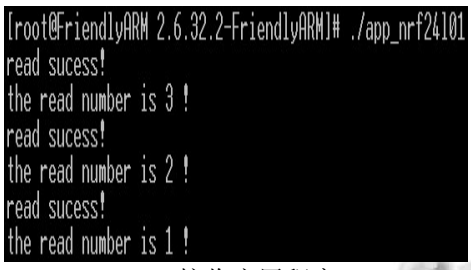
巡检方式 :

```
main（）
{   int i,nrf24l01_fd，receive[32];
nrf24l01_fd=open("/dev/nrf24l01",O_RDWR);     //以可读可写的方式打开设备文件
   while(1)
{ i=ioctl(nrf24l01_fd,0,0); //调用 ioctl 函数查询标志寄存器状态
if（i==3）
read(nrf24l01_fd,receive,1); //寄存器状态表明接收到有效数据则调用 read 函数
.....}
close( nrf24l01_fd); }
```

以上程序在 linux 平台上的实验效果截图如图 1 所示：




(a) 发送应用程序




(b) 接收应用程序

图 1 实验效果截图

以发送“321”这个数字队列为例，从图 1 可知发送和接收程序均较好地实现了功能。

## 5 总结

本文结合 linux 系统字符驱动的基本框架，详细论述了开发无线通讯芯片 nRF24L01 驱动的过程和方法，对开发其他类型的字符驱动也具有一定的借鉴价值。

本文的创新点是在驱动中采用了中断和巡检两种方法来处理无线数据的收发情况，以适应于不同的应用环境，并得到了验证。目前该驱动已成功地运用在本实验中心对气体传感器信号的无线监测项目当中。

## 参考文献

1 时志云,盖建平.新型高速无线射频器件 nRF24L01 及其应用.国外电子元器件,2007,(8):42–44.

2 陈莉君.深入理解 Linux 内核源代码.北京:人民邮电出版社,2002.

3 李俊.嵌入式 Linux 设备驱动开发详解.北京:人民邮电出版社,2008.

4 李桦,高飞,等.嵌入式 Linux 设备驱动程序研究.微计算机信息,2010,5(2):68–70.

5 Rubini A, Corbet J. Linux Device Drivers. 2nd Ed. O’Reilly Media, Inc.June. 2001.

(上接第 237 页)

of materialized views and indexes for SQL databases. Proceedings of 26th International Conference on Very Large Data Bases, Cairo, 2000. San Francisco: Morgan Kaufmann Publishers Inc, 2000. 496–505.

10 Ladjel B, Kamel B. An evolutionary approach to schema partitioning selection in a data warehouse. Proc. of 7th International Conference on Data Warehousing and Knowledge Discovery. Copenhagen, 2005. Heidelberg: Springer Berlin, 2005. 115–125.

11 Bellatreche M, Drias N. Selection and pruning algorithms for bitmap index selection problem using data mining. 9th International Conference on Data Warehousing and Knowledge Discovery, Regensburg, 2007. 221–230.